## Handling requests with the middleware pipeline

**Keywords:**

"middleware are C# classes or functions that handle an HTTP request or response"

"chained together, with the output of one middleware acting as the input to the next middleware"

"the middleware pipeline is one of the most important parts of configuration"

"understanding how to build and compose middleware is key to adding functionality to your applications"

"each component adding a discrete piece of functionality"

"how to arrange them in the correct way for your application"

"middleware component often maps to a 'cross-cutting concern'"

"logging, error handling, and security are classic cross-cutting concerns"

"all requests pass through the middleware pipeline"

"compose individual middleware components into a pipeline"

"build a simple static-file server that returns requested files from a folder on disk"

"importance of ordering in the middleware pipeline"

"errors are a fact of life for all applications"

"classic crosscutting concerns"

"handle exceptions and errors using middleware provided by Microsoft"

"DeveloperExceptionPageMiddleware—Provides quick error feedback"

"ExceptionHandlerMiddleware—Provides a user-friendly generic error page in production"

"StatusCodePagesMiddleware—Converts raw error status codes into user-friendly error pages"

"middleware is well placed to provide the required functionality"

"you can go a long way using the components provided as part of ASP.NET Core"

**Questions:**

Q: What are middleware in ASP.NET Core?

A: Middleware are C# classes or functions that handle an HTTP request or response.

Q: How are middleware components connected in ASP.NET Core?

A: They are chained together, with the output of one middleware acting as the input to the next middleware.

Q: Why is the middleware pipeline important?

A: It is one of the most important parts of configuration for defining how your application behaves and how it responds to requests.

Q: What is key to adding functionality to your applications?

A: Understanding how to build and compose middleware.

Q: What does each middleware component add to the pipeline?

A: Each component adds a discrete piece of functionality.

Q: What type of concerns do middleware components often map to?

A: Middleware components often map to a “cross-cutting concern.”

Q: What are examples of classic cross-cutting concerns?

A: Logging, error handling, and security.

Q: Why is the middleware pipeline a preferred location for handling cross-cutting concerns?

A: Because all requests pass through the middleware pipeline.

Q: What will you learn to build in section 3.2?

A: A simple static-file server that returns requested files from a folder on disk.

Q: What aspect of the middleware pipeline is explored when using multiple middleware?

A: The importance of ordering in the middleware pipeline.

Q: Why is error handling important in application building?

A: Because errors are a fact of life for all applications.

Q: How can you handle exceptions and errors using middleware?

A: Using middleware provided by Microsoft.

Q: What does DeveloperExceptionPageMiddleware provide?

A: Quick error feedback when building an application.

Q: What does ExceptionHandlerMiddleware provide?

A: A user-friendly generic error page in production.

Q: What does StatusCodePagesMiddleware do?

A: Converts raw error status codes into user-friendly error pages.

Q: What is the advantage of combining middleware for error handling?

A: You can ensure that any errors won’t leak security details and won’t break your app.

Q: What will you not learn in this chapter?

A: You won’t see how to build your own middleware.

Q: What makes it easier to understand custom middleware requirements?

A: Understanding the middleware pipeline and its behavior.

**What is middleware?**

**Keywords:**

"middleware are C# classes that can handle an HTTP request or response"

"handle an incoming HTTP request by generating an HTTP response"

"process an incoming HTTP request, modify it, and pass it on to another piece of middleware"

"process an outgoing HTTP response, modify it, and pass it on"

"a piece of logging middleware might note when a request arrived"

"image-resizing middleware component might spot an incoming request for an image"

"the most important piece of middleware in most ASP.NET Core applications is the EndpointMiddleware class"

"this arrangement... is referred to as a pipeline"

"cross-cutting concerns of your application"

"logging each request"

"adding standard security headers to the response"

"associating a request with the relevant user"

"setting the language for the current request"

"the authentication middleware associates the request with a user"

"the authorization middleware uses this detail to verify whether the user has permission"

"the authorization middleware can short-circuit the pipeline"

"the pipeline is bidirectional"

"you define the middleware pipeline in code as part of your initial application configuration in Startup"

"middleware is the fundamental source of behavior in your application"

"requests are passed to the middleware pipeline as HttpContext objects"

"middleware pipeline as being a series of concentric components"

**Questions:**

Q: What are middleware in ASP.NET Core?

A: Middleware are C# classes that can handle an HTTP request or response.

Q: What can middleware do with an incoming HTTP request?

A: Handle an incoming HTTP request by generating an HTTP response.

Q: What can middleware do with an outgoing HTTP response?

A: Process an outgoing HTTP response, modify it, and pass it on.

Q: What does a piece of logging middleware do?

A: Note when a request arrived and then pass it on to another piece of middleware.

Q: What does the image-resizing middleware component do?

A: Spot an incoming request for an image with a specified size, generate the requested image, and send it back to the user.

Q: What is the most important piece of middleware in most ASP.NET Core applications?

A: The EndpointMiddleware class.

Q: What does the EndpointMiddleware class normally generate?

A: All your HTML pages and API responses.

Q: What is the arrangement of calling one middleware from another referred to as?

A: A pipeline.

Q: What is one of the most common use cases for middleware?

A: The cross-cutting concerns of your application.

Q: What are examples of cross-cutting concerns?

A: Logging each request, adding standard security headers to the response, associating a request with the relevant user, setting the language for the current request.

Q: What does the authentication middleware do?

A: Associates the request with a user.

Q: What does the authorization middleware use to verify permissions?

A: The detail added by the authentication middleware.

Q: What happens if the user doesn't have permission?

A: The authorization middleware can short-circuit the pipeline, generating a response directly.

Q: What is a key point about the middleware pipeline's direction?

A: The pipeline is bidirectional.

Q: How is the middleware pipeline defined?

A: In code as part of your initial application configuration in Startup.

Q: What is middleware the fundamental source of?

A: Behavior in your application.

Q: In what form are requests passed to the middleware pipeline?

A: As HttpContext objects.

Q: What builds an HttpContext object from an incoming request?

A: The ASP.NET Core web server.

Q: What metaphor is used to describe the middleware pipeline's structure?

A: A series of concentric components, similar to a traditional matryoshka (Russian) doll.

## Combining middleware in a pipeline

**Keywords:**

"each middleware component has a single primary concern"

"logging middleware will only deal with logging the request"

"authentication middleware is only concerned with identifying the current user"

"static-file middleware is only concerned with returning static files"

"each of these concerns is highly focused"

"components themselves small and easy to reason about"

"your app added flexibility"

"adding a static-file middleware doesn't mean you're forced into having image-resizing behavior or authentication"

"compose multiple middleware components together into a pipeline"

"each middleware has access to the original request"

"any changes made by previous middleware in the pipeline"

"each middleware can inspect and/or modify the response"

"build complex application behaviors from small, focused components"

"create a middleware pipeline by composing small components together"

"create a holding page and to serve static files from a folder on disk"

"decompose it to understand why it's built like it is"

**Questions:**

Q: What does each middleware component generally have?

A: A single primary concern.

Q: What is the function of logging middleware?

A: Logging middleware will only deal with logging the request.

Q: What is authentication middleware concerned with?

A: Identifying the current user.

Q: What is static-file middleware responsible for?

A: Returning static files.

Q: What makes the components small and easy to reason about?

A: Each of these concerns is highly focused.

Q: What flexibility does adding static-file middleware provide?

A: Adding a static-file middleware doesn't mean you're forced into having image-resizing behavior or authentication.

Q: How do you build a complete application using middleware?

A: You compose multiple middleware components together into a pipeline.

Q: What does each middleware have access to in the pipeline?

A: The original request, plus any changes made by previous middleware in the pipeline.

Q: What can middleware do once a response has been generated?

A: Inspect and/or modify the response as it passes back through the pipeline.

Q: What does composing small, focused components allow you to build?

A: Complex application behaviors.

Q: What will you learn in the rest of this section?

A: How to create a middleware pipeline by composing small components together.

Q: What will you use standard middleware components to create?

A: A holding page and to serve static files from a folder on disk.

Q: What will you do with the default middleware pipeline from chapter 2?

A: Decompose it to understand why it's built like it is.

## Simple pipeline scenario 1: A holding page

**Keywords:**

"Simple pipeline scenario 1: A holding page"

"create an app consisting of a holding page"

"useful when you're first setting up your application"

"ASP.NET Core framework is composed of many small, individual libraries"

"add a piece of middleware by referencing a package in your application's .csproj project file"

"configuring the middleware in the Configure method of your Startup class"

"Microsoft ships many standard middleware components with ASP.NET Core"

"you can also use third-party components from NuGet and GitHub"

"build your own custom middleware"

"view the source code for all the middleware that comes as part of ASP.NET Core"

"most of the middleware in the src/Middleware folder"

"authentication and authorization middleware can be found in the src/Security folder"

"WelcomePageMiddleware is designed to quickly provide a sample page"

"you wouldn't use it in a production app"

"a single, self-contained middleware component"

"WelcomePageMiddleware is included as part of the base ASP.NET Core framework"

"the request passes to the ASP.NET Core web server"

"builds a representation of the request and passes it to the middleware pipeline"

"WelcomePageMiddleware receives the request and must decide how to handle it"

"generating an HTML response"

"response passes back to the ASP.NET Core web server"

**Questions:**

Q: What does the first simple pipeline scenario consist of?

A: A holding page.

Q: When is a holding page useful?

A: When you're first setting up your application.

Q: How do you typically add a piece of middleware?

A: By referencing a package in your application's .csproj project file and configuring the middleware in the Configure method of your Startup class.

Q: Where can you view the application code for the book?

A: In the GitHub repository at [https://github.com/andrewlock/asp-dot-net-core-in-action-2e](https://github.com/andrewlock/asp-dot-net-core-in-action-2e).

Q: What does the ASP.NET Core framework consist of?

A: Many small, individual libraries.

Q: Where can you find the source code for the middleware that comes with ASP.NET Core?

A: In the main ASP.NET Core GitHub repository.

Q: Where is most of the middleware located in the GitHub repository?

A: In the src/Middleware folder.

Q: Where can the authentication and authorization middleware be found?

A: In the src/Security folder.

Q: What is WelcomePageMiddleware designed to do?

A: Quickly provide a sample page when you're first developing an application.

Q: Why wouldn't you use WelcomePageMiddleware in a production app?

A: Because you can't customize the output.

Q: Is WelcomePageMiddleware included in the base ASP.NET Core framework?

A: Yes, so you don't need to add a reference to any additional NuGet packages.

Q: What happens when the application receives an HTTP request?

A: The request passes to the ASP.NET Core web server, which builds a representation of the request and passes it to the middleware pipeline.

Q: What does WelcomePageMiddleware do with the request?

A: It receives the request and must decide how to handle it.

Q: What does WelcomePageMiddleware generate in response to any request?

A: An HTML response.

Q: Where does the response go after being generated?

A: It passes back to the ASP.NET Core web server, which forwards it on to the user.

**Sending Broadcast and Follow-Up Emails**

**Keywords:**

"define the middleware pipeline in the Configure method of Startup"

"adding middleware to an IApplicationBuilder object"

"single middleware component"

"Startup class for this application is very simple"

"no configuration and no services"

"call UseWelcomePage"

"calling methods on IApplicationBuilder"

"these are extension methods"

"add functionality to the IApplicationBuilder class"

"UseWelcomePage method adds the WelcomePageMiddleware"

"UseMiddleware<WelcomePageMiddleware>()"

"extension method for each piece of middleware"

"starting the method name with Use"

"improve discoverability when adding middleware"

"middleware as part of the core framework"

"IntelliSense in Visual Studio and other IDEs"

"order in which you make calls to IApplicationBuilder in Configure defines the order"

"take care when adding middleware to the pipeline"

"access data created by middleware that comes before it"

"returning the same response no matter which URL you navigate to"

"define your application behavior using middleware"

**Questions:**

Q: Where do you define the middleware pipeline in an ASP.NET Core application?

A: In the Configure method of Startup.

Q: What object do you add middleware to when defining the pipeline?

A: An IApplicationBuilder object.

Q: What is the only required method in a simple Startup class with no configuration or services?

A: Configure.

Q: What method is called to add WelcomePageMiddleware in the pipeline?

A: UseWelcomePage.

Q: What kind of methods are used to add middleware to IApplicationBuilder?

A: Extension methods.

Q: Why are extension methods used with IApplicationBuilder?

A: To add functionality while keeping implementations isolated from it.

Q: What does UseWelcomePage call behind the scenes?

A: UseMiddleware<WelcomePageMiddleware>().

Q: Why do extension method names for middleware typically start with "Use"?

A: To improve discoverability when adding middleware to your application.

Q: What helps you view all the middleware available in ASP.NET Core?

A: IntelliSense in Visual Studio and other IDEs.

Q: What does calling UseWelcomePage do in the pipeline?

A: Adds the WelcomePageMiddleware as the next middleware in the pipeline.

Q: What determines the order in which middleware runs?

A: The order in which you make calls to IApplicationBuilder in Configure.

Q: Why must you take care when adding middleware to the pipeline?

A: A component can only access data created by middleware that comes before it.

Q: What kind of response does the basic application return?

A: The same response no matter which URL you navigate to.

Q: What does the basic example demonstrate about middleware?

A: How easy it is to define your application behavior using middleware.

## Simple pipeline scenario 2: Handling static files

**Keywords:**

"one of the simplest middleware pipelines"

"static-file application"

"serve a number of pages using static files"

"saved to disk during development"

"StaticFileMiddleware to create an application"

"serves static files from the wwwroot folder"

"image called moon.jpg exists in the wwwroot folder"

"404 HTTP error code response"

"browser...might see a completely blank page"

"single piece of middleware, StaticFileMiddleware"

"UseStaticFiles is all that’s required"

"StaticFileMiddleware receives the request"

"middleware handles the request and returns the file"

"request effectively passes through the static-file middleware unchanged"

"ASP.NET Core automatically adds a 'dummy' piece of middleware"

"pipeline will automatically return a simple 404 error response"

"behavior of the ASP.NET Core middleware pipeline"

"static-file middleware in particular"

"static files will form one part of your middleware pipeline"

"combine multiple middleware"

**Questions:**

Q: What type of application is created using one of the simplest middleware pipelines?

A: A static-file application.

Q: What types of content are normally saved to disk and served as static files?

A: Images, JavaScript, and CSS stylesheets.

Q: Which middleware is used to serve static files from the wwwroot folder?

A: StaticFileMiddleware.

Q: What happens when you request /moon.jpg and it exists in the wwwroot folder?

A: It’s loaded and returned as the response to the request.

Q: What response is sent if a requested file doesn't exist in the wwwroot folder?

A: A 404 HTTP error code response.

Q: What might Internet Explorer display when a missing file is requested?

A: A completely blank page.

Q: How many middleware components are used to build this application's pipeline?

A: A single piece of middleware, StaticFileMiddleware.

Q: What method is used to configure the middleware pipeline for serving static files?

A: UseStaticFiles.

Q: What does StaticFileMiddleware do if the requested file exists?

A: Handles the request and returns the file as the response.

Q: What happens if the requested file doesn't exist and there's only one middleware?

A: The request effectively passes through the static-file middleware unchanged.

Q: What does ASP.NET Core add to the end of the pipeline by default?

A: A "dummy" piece of middleware.

Q: What does the dummy middleware always return if it's called?

A: A 404 response.

Q: What is the default response if no middleware generates a response for a request?

A: A simple 404 error response to the browser.

Q: What does this basic application demonstrate clearly?

A: The behavior of the ASP.NET Core middleware pipeline and the static-file middleware in particular.

Q: How are static files typically used in more complex applications?

A: As one part of your middleware pipeline.

## Simple pipeline scenario 3: A Razor Pages application

**Keywords:**

"middleware pipeline"

"defines your application's behavior"

"combine several standard middleware components"

"Configure method of Startup"

"basic middleware pipeline"

"ASP.NET Core Razor Pages template"

"navigate to the home page of the application"

"four pieces of middleware"

"routing middleware to choose a Razor Page to execute"

"endpoint middleware to generate the HTML from a Razor Page"

"static-file middleware to serve the CSS and image files"

"exception handler middleware to handle any errors"

"configuration of the middleware pipeline"

"call to AddRazorPages() in ConfigureServices"

"required when using Razor Pages"

**Questions:**

Q: What does the middleware pipeline define in an ASP.NET Core application?

A: It defines your application's behavior.

Q: Where do you add middleware to form the pipeline in ASP.NET Core?

A: In the Configure method of Startup.

Q: What kind of middleware pipeline do you begin with in this section?

A: A basic middleware pipeline that you'd find in a typical ASP.NET Core Razor Pages template.

Q: What is shown in figure 3.10?

A: The output when you navigate to the home page of the application.

Q: How many pieces of middleware are required to create the application?

A: Four pieces of middleware.

Q: What is the purpose of the routing middleware?

A: To choose a Razor Page to execute.

Q: What does the endpoint middleware do?

A: It generates the HTML from a Razor Page.

Q: What is the role of the static-file middleware?

A: To serve the CSS and image files from the wwwroot folder.

Q: Why is exception handler middleware used?

A: To handle any errors that might occur.

Q: What method must be called in ConfigureServices when using Razor Pages?

A: AddRazorPages().

**A basic middleware pipeline for a Razor Pages application**

**Keywords:**

"addition of middleware to IApplicationBuilder"

"all the methods for adding middleware start with Use"

"convention of using extension methods"

"easier to discover"

"order of the Use methods in the Configure method"

"order in which they’re added to the pipeline"

"exception handler middleware is called first"

"static-file middleware"

"routing middleware selects a Razor Page"

"endpoint middleware executes the selected Razor Page"

"automatic dummy middleware returns a 404 response"

"routing and endpoint middleware were combined"

"insert middleware between the routing and endpoint middleware"

"both listening for the same path"

"WelcomePageMiddleware"

"respond only to the '/' path"

**Questions:**

Q: What prefix is used for all the methods that add middleware?

A: Use.

Q: Why are middleware methods prefixed with "Use"?

A: To make them easier to discover.

Q: What determines the order in which middleware is added to the pipeline?

A: The order of the Use methods in the Configure method.

Q: Which middleware is called first in the example pipeline?

A: The exception handler middleware.

Q: What does the static-file middleware do if a requested file exists?

A: It will generate a response.

Q: What happens if the static-file middleware cannot handle the request?

A: It passes the request on to the routing middleware.

Q: What does the routing middleware do with the request URL?

A: It selects a Razor Page based on the request URL.

Q: What middleware executes the selected Razor Page?

A: The endpoint middleware.

Q: What happens if no Razor Page can handle the requested URL?

A: The automatic dummy middleware returns a 404 response.

Q: What was the routing and endpoint middleware combined into in earlier ASP.NET Core versions?

A: A single “MVC” middleware.

Q: What does splitting routing and endpoint middleware allow?

A: To insert middleware between the routing and endpoint middleware.

Q: What happens when two pieces of middleware listen for the same path?

A: The impact of ordering can most obviously be seen.

Q: Which middleware is configured in the example to respond to the "/" path?

A: WelcomePageMiddleware.

Q: What kind of response does WelcomePageMiddleware return?

A: A fixed HTML response.

## Adding WelcomePageMiddleware to the pipeline

**Keywords:**

"WelcomePageMiddleware is earlier in the pipeline"

"short-circuiting the pipeline"

"none has an opportunity to generate a response"

"moved WelcomePageMiddleware to the end of the pipeline"

"you'd never see the Welcome page"

"always consider the order of middleware"

"middleware added earlier in the pipeline will run"

"middleware pipeline is bidirectional"

"handle both the incoming request and the outgoing response"

"order of middleware is most important"

"ExceptionHandlerMiddleware at the start"

"error handling middleware characteristically ignores the incoming request"

"modifying it when an error has occurred"

"types of error handling middleware"

**Questions:**

Q: Why does WelcomePageMiddleware return a response when it receives a request to "/"?

A: Because it is earlier in the pipeline.

Q: What happens to the other middleware when WelcomePageMiddleware handles the "/" request?

A: None of the other middleware in the pipeline runs for the request.

Q: What would happen if WelcomePageMiddleware is moved after the call to UseEndpoints?

A: You'd never see the Welcome page.

Q: What should developers always consider when adding middleware to the Configure method?

A: The order of middleware.

Q: What is true about middleware added earlier in the pipeline?

A: It will run before middleware added later.

Q: How does the middleware pipeline operate directionally?

A: The middleware pipeline is bidirectional.

Q: What does each middleware component get an opportunity to handle?

A: Both the incoming request and the outgoing response.

Q: For which type of components is the order of middleware most important?

A: Those components that create or modify the outgoing response.

Q: Where was ExceptionHandlerMiddleware placed in the previous example?

A: At the start of the application’s middleware pipeline.

Q: What does error handling middleware typically ignore?

A: The incoming request.

Q: When does error handling middleware modify the response?

A: Only when an error has occurred.

Q: What topic will be detailed in the next section?

A: The types of error handling middleware that are available to use.

## Handling errors using middleware

**Keywords:**

"errors are a fact of life"

"handles these errors gracefully"

"your whole application to fail"

"every feature is opt-in"

"you need to explicitly enable it"

"exceptions and error status codes"

"NullReferenceException"

"exception occurs in a middleware component"

"web server will return a 500 status code"

"middleware might generate an error status code"

"pipeline will return a 404 error"

"generic, unfriendly page"

"error handling middleware attempts to address these problems"

"returns details of the error"

"returns a generic, but friendly, HTML page"

"place error handling middleware early in the middleware pipeline"

"catch any errors generated in subsequent middleware"

"responses generated by middleware earlier in the pipeline can’t be intercepted"

"available as part of the base ASP.NET Core framework"

"don’t need to reference any additional NuGet packages"

**Questions:**

Q: What happens to an application if errors are not handled gracefully?

A: It can cause your whole application to fail.

Q: How is error handling treated in ASP.NET Core’s design philosophy?

A: Every feature is opt-in, so you need to explicitly enable error handling.

Q: What are the two types of errors focused on in this section?

A: Exceptions and error status codes.

Q: What is an example of a common exception mentioned in the passage?

A: NullReferenceException.

Q: What happens if an exception in a middleware component is not handled in the pipeline?

A: The web server will return a 500 status code back to the user.

Q: What error status code is returned when a requested path isn't handled?

A: A 404 error.

Q: What does error handling middleware typically do before the app returns a response?

A: It modifies the response to return error details or a friendly HTML page.

Q: Where should error handling middleware be placed in the middleware pipeline?

A: Early in the middleware pipeline.

Q: Why must error handling middleware be placed early in the pipeline?

A: To catch any errors generated in subsequent middleware.

Q: Can responses generated by middleware earlier than error handling middleware be intercepted?

A: No, they can't be intercepted.

Q: Are additional NuGet packages required to use error handling middleware in ASP.NET Core?

A: No, they are available as part of the base ASP.NET Core framework.

## Viewing exceptions in development: DeveloperExceptionPage

**Keywords:**

"DeveloperExceptionPageMiddleware"

"app.UseDeveloperExceptionPage()"

"generates a friendly HTML page"

"returns with a 500 status code"

"exception stack trace"

"source code at the line the exception occurred"

"details of the request"

"cookies or headers that had been sent"

"invaluable for debugging a problem"

"represent a security risk if used incorrectly"

"never return more details about your application to users than absolutely necessary"

"only ever use DeveloperExceptionPage when developing your application"

"Never use the developer exception page when running in production"

"security risk"

"publicly reveal details about your application’s code"

"another general-purpose error handling middleware you can use in production"

"ExceptionHandlerMiddleware"

**Questions:**

Q: What middleware does Microsoft provide to give detailed error information during development?

A: DeveloperExceptionPageMiddleware.

Q: How is DeveloperExceptionPageMiddleware added to the middleware pipeline?

A: Using app.UseDeveloperExceptionPage().

Q: What does DeveloperExceptionPageMiddleware return to the user when an exception occurs?

A: A friendly HTML page with a 500 status code.

Q: What kind of details does the developer exception page contain?

A: The exception stack trace, the source code at the line the exception occurred, and details of the request such as cookies or headers.

Q: Why is it important to limit the details returned to users by DeveloperExceptionPageMiddleware?

A: Because returning too many details represents a security risk.

Q: When should you use DeveloperExceptionPageMiddleware?

A: Only when developing your application.

Q: What is the warning given about using DeveloperExceptionPage in production?

A: Never use the developer exception page when running in production as it is a security risk.

Q: What could happen if you use the developer exception page in production?

A: It could publicly reveal details about your application’s code, making you an easy target for attackers.

Q: What middleware should you use instead of DeveloperExceptionPageMiddleware in production?

A: ExceptionHandlerMiddleware.

## Handling exceptions in production: ExceptionHandlerMiddleware

**Keywords:**

"developer exception page is handy when developing your applications"

"shouldn’t use it in production"

"can leak information about your app to potential attackers"

"users will see unfriendly error pages or blank pages"

"using ExceptionHandlerMiddleware"

"user will be presented with a custom error page"

"custom error page ... provides the necessary details about the error"

"keep the look and feel of the application"

"displaying the currently logged-in user"

"displaying an appropriate message to the user instead of the full details of the exception"

"developer exception page used in combination with ExceptionHandlerMiddleware"

"similar manner to that shown in listing 3.5"

**Questions:**

Q: Why shouldn’t you use the developer exception page in production?

A: Because it can leak information about your app to potential attackers.

Q: What problem occurs if you do not catch errors in your application?

A: Users will see unfriendly error pages or blank pages, depending on the browser.

Q: How does ExceptionHandlerMiddleware improve the user experience when an error occurs?

A: It presents the user with a custom error page consistent with the application that only provides necessary error details.

Q: What features can a custom error page include to maintain the application’s look and feel?

A: Using the same header, displaying the currently logged-in user, and showing an appropriate message instead of full exception details.

Q: How are the developer exception page and ExceptionHandlerMiddleware typically used together?

A: They are used in combination in the Configure method of almost any ASP.NET Core application.

## Configuring exception handling for development and production

**Keywords:**

"configure different middleware pipelines depending on the environment"

"vary your pipeline based on other values, such as settings loaded from configuration"

"ExceptionHandlerMiddleware will invoke this path after it captures an exception"

"re-execute a middleware pipeline in order to generate the response"

"sequence of events when an exception occurs somewhere in the middleware pipeline"

"middleware throws an exception"

"ExceptionHandlerMiddleware catches the exception"

"middleware overwrites the request path with the provided error handling path"

"middleware pipeline generates a new response as normal"

"modifies the status code to a 500 error"

"re-executing the pipeline brings the ability to have your error messages integrated into your normal site layout"

"menu bar with dynamically generated links or display the current user's name"

"subsequent middleware will treat the re-execution as a new request"

"middleware can only modify a response generated further down the pipeline if the response hasn't yet been sent to the client"

"error occurs while ASP.NET Core is sending a static file to a client"

"error handling path throws an error during the re-execution of the pipeline"

"ExceptionHandlerMiddleware has already tried to intercept a request"

"web server returns a raw 500 error"

"good practice to make your error handling pages as simple as possible"

"use a static error page that will always work"

"ExceptionHandlerMiddleware and DeveloperExceptionPageMiddleware are great for catching exceptions"

"exceptions aren't the only sort of errors you'll encounter"

"handle both exceptions and error status codes to provide a coherent user experience"

**Questions:**

Q: What does ExceptionHandlerMiddleware do after it captures an exception?

A: It invokes the provided error handling path to generate the final response.

Q: Why is re-executing the middleware pipeline beneficial for error handling?

A: It allows error messages to be integrated into the normal site layout with dynamic content like menus and usernames.

Q: What happens if the error handling path itself throws an error during re-execution?

A: ExceptionHandlerMiddleware lets the error propagate to the top, and the web server returns a raw 500 error.

Q: Why is it often better to use a static error page rather than a dynamic one?

A: Because a static page will always work and reduces the risk of errors occurring during error handling.

Q: Under what condition can middleware not modify a response further down the pipeline?

A: If the response has already begun to be sent to the client.

Q: Can you configure different middleware pipelines based on environment or configuration settings?

A: Yes, it is perfectly acceptable to configure different middleware pipelines depending on the environment or other values such as configuration.

Q: What is a recommended practice for designing error handling pages?

A: To make them as simple as possible to reduce the possibility of errors occurring.

Q: Besides exceptions, what other types of errors should you handle in your middleware pipeline?

A: HTTP error status codes.

## Handling other errors: StatusCodePagesMiddleware

**Keywords:**

"500 'server error' is sent when an exception occurs and isn't handled"

"404 'file not found' error is sent when a URL isn't handled by any middleware"

"404 errors are often used to indicate that a specific requested object was not found"

"handle these error codes and return an error page that's in keeping with the rest of your application"

"Microsoft provides StatusCodePagesMiddleware for handling this use case"

"add it early in your middleware pipeline"

"app.UseStatusCodePages();"

"intercept any response that has an HTTP status code that starts with 4xx or 5xx and has no response body"

"add a plain text response body, indicating the type and name of the response"

"re-execute the pipeline when an error is captured"

"app.UseStatusCodePagesWithReExecute("/{0}");"

"the middleware will replace this token with the status code number"

"handler for the path (typically a Razor Page) has access to the status code and can optionally tailor the response"

"create different error pages for different error codes"

"error pages are consistent with the rest of your application"

"StatusCodePagesMiddleware will only modify the response if no response body has been written"

"StatusCodePagesMiddleware has additional overloads that let you execute custom middleware when an error occurs"

"you may not always want your error handling middleware to generate HTML pages"

**Questions:**

Q: What HTTP status code is sent when an exception occurs and isn't handled?

A: A 500 "server error" is sent when an exception occurs and isn't handled.

Q: What does a 404 error indicate besides an unhandled URL?

A: It often indicates that a specific requested object was not found.

Q: What middleware does Microsoft provide to handle HTTP status code errors like 4xx and 5xx?

A: Microsoft provides StatusCodePagesMiddleware for handling this use case.

Q: When should StatusCodePagesMiddleware be added to the middleware pipeline?

A: It should be added early in the middleware pipeline.

Q: What does app.UseStatusCodePages() do without additional configuration?

A: It intercepts any response with a 4xx or 5xx status code and no response body and adds a plain text response body indicating the type and name of the response.

Q: How can you configure StatusCodePagesMiddleware to re-execute the pipeline for error handling?

A: By using app.UseStatusCodePagesWithReExecute("/{0}");

Q: What does the "{0}" token in the error handling path represent?

A: The token is replaced with the status code number when the path is re-executed.

Q: What advantage does re-executing the pipeline with StatusCodePagesMiddleware provide?

A: It allows creating different error pages that are consistent with the rest of the application and tailored to specific error codes.

Q: Will StatusCodePagesMiddleware modify a response if another middleware has already written a response body?

A: No, it will only modify the response if no response body has been written.

Q: What other option does StatusCodePagesMiddleware offer besides re-executing a Razor Pages path?

A: It has additional overloads that let you execute custom middleware when an error occurs.

## Error handling middleware and Web APIs

**Keywords:**

"ASP.NET Core isn't only great for creating user-facing web applications"

"creating HTTP services that can be accessed from another server application, from a mobile app, or from a user's browser"

"you probably won't be returning HTML to the client, but rather XML or JSON"

"HTTP 500 status code and a JSON body describing the error is more useful to a consuming application"

"ASP.NET Core allows you to do exactly this when you create Web API controllers"

"how to use and compose middleware to form a pipeline"

"how to handle errors in your application"

"build your own custom middleware"

"perform complex operations on the middleware pipeline, such as forking it in response to specific requests"

"Razor Pages, and at how they can be used to build websites"

"MVC design pattern, its relationship with Razor Pages in ASP.NET Core, and when to choose one approach over the other"

**Questions:**

Q: Besides user-facing web applications, what else is ASP.NET Core great for creating?

A: ASP.NET Core is also great for creating HTTP services that can be accessed from another server application, from a mobile app, or from a user's browser.

Q: What formats might ASP.NET Core return instead of HTML when creating HTTP services?

A: It might return XML or JSON instead of HTML.

Q: Why is returning an HTML error page problematic for applications expecting JSON?

A: Returning an HTML page to an application expecting JSON could easily break it unexpectedly.

Q: What is more useful to a consuming application when an error occurs in an HTTP service?

A: The HTTP 500 status code and a JSON body describing the error is more useful.

Q: When creating HTTP services in ASP.NET Core, how can you return JSON error responses?

A: ASP.NET Core allows you to return JSON error responses when you create Web API controllers.

Q: What middleware topics have been covered so far according to the passage?

A: How to use and compose middleware to form a pipeline and how to handle errors in your application.

Q: What will you learn about in the next chapter according to the passage?

A: The next chapter will cover Razor Pages, how they can be used to build websites, the MVC design pattern, and its relationship with Razor Pages in ASP.NET Core.

Q: What advanced middleware topics will be covered later?

A: Building custom middleware and performing complex operations on the middleware pipeline, such as forking it in response to specific requests.